A quantum architecture for multiplying signed integers

To cite this article: J J Álvarez-Sánchez et al 2008 J. Phys.: Conf. Ser. 128 012013

View the article online for updates and enhancements.

Related content
- Reflections on the information paradigm in quantum and gravitational physics
  Philipp Andres Höhn
- A dual-purpose glass blowing torch
  D. Johnston
- Recursion method for binary intermetallic compounds
  U H Glaser and P Rennert

Recent citations
- Quantum arithmetic with the quantum Fourier transform
  Lidia Ruiz-Perez and Juan Carlos Garcia-Escartin
- Quantum Fourier transform in computational basis
  S. S. Zhou et al
- Cost-efficient design of a quantum multiplier–accumulator unit
  Hafiz Md. Hasan Babu
A quantum architecture for multiplying signed integers

J J Álvarez-Sánchez¹, J V Álvarez-Bravo¹ and L M Nieto²
¹ Departamento de Informática, E.U. de Informática, Universidad de Valladolid, 40005 Segovia, Spain
² Departamento de Física Teórica, Atómica y Óptica, Universidad de Valladolid, 47071 Valladolid, Spain
E-mail: jjalvarez@infor.uva.es, jvalvarez@infor.uva.es, luismi@metodos.fam.cie.uva.es

Abstract. A new quantum architecture for multiplying signed integers is presented based on Booth’s algorithm, which is well known in classical computation. It is shown how a quantum binary chain might be encoded by its flank changes, giving the final product in 2’s-complement representation.

1. Introduction and motivation
Factoring big numbers in polynomial time, searching for an item in an unstructured data base faster than any classical (not quantum) algorithm, or achieving quantum cryptographic protocols for a quantum public key distribution that gives us better eavesdropping detection techniques, are some of the spectacular theoretical results [1, 2, 3] that have supported the idea of building a quantum computer or, at least, a quantum microprocessor for performing some specific tasks. In this context, a fundamental issue that has to be considered is how could we work out arithmetic calculus by means of quantum architectures.

Indeed, the first step for performing complex quantum arithmetic calculus is to build a quantum architecture which will be able to add two positive integer numbers (a ‘quantum adder’). This quantum algorithm has been the target of many quantum arithmetic researchers, who can be divided in two groups: those who carry out the quantization of classical arithmetic algorithms [4, 5, 6] (who, indeed, rewrite classical arithmetical algorithms, that are irreversible, as classical reversible ones), and those who use the Quantum Fourier Transform for building adders [7].

In this paper the first approach is chosen in order to design a quantum circuit that will be able to reproduce the outcomes of a classical arithmetic circuit for multiplying signed integers. The keystone is to avoid the construction of different circuits for different arithmetic tasks. We could consider, for instance, the subtraction of two positive numbers to illustrate the point of view we are assuming. In principle, we could tackle the problem by just applying the corresponding subtraction truth table. Nevertheless, if we adopt this point of view, hardware different from the one used for addition will necessary to perform the subtraction operation.

This work is organized as follows. In section 2 some relevant aspects of classical and quantum computer arithmetics are exposed. In section 3, a brief and basic description of the classical
Booth algorithm for multiplying signed integers is given. Then, the quantum counterpart of Booth’s algorithm is explicitly constructed in section 4, which is the central part of this work. We end with a discussion of the results and some indications of future work.

2. Classical and quantum computer arithmetics
Let us start by remembering that, in a classical computer, the arithmetic calculus is performed by the arithmetic logic unit, that is a part of the *brain* of the computer: the central processing unit (CPU). Within the arithmetic logic unit the fundamental arithmetic operations (addition, subtraction, multiplication and division) are carried out over the numbers, which are represented using Boolean algebra. We can assume that the same will be true in the quantum context, using quantum bits (‘qubits’). Either in the classical or in the quantum case, any integer Boolean number can be written as

\[ p = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \ldots + x_02^0 \]  

(1)

with \( x_i = 0, 1 \), and the algebraic internal operations (+,·), usually called **OR** and **AND**, are defined by the truth tables shown in table 1.

<table>
<thead>
<tr>
<th>( x_i )</th>
<th>( y_i )</th>
<th>( x_i \cdot y_i )</th>
<th>( x_i + y_i )</th>
<th>( x_i \oplus y_i )</th>
<th>( c_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Thus, one can introduce, in an obvious way, the addition of two integers just adding the bits that have the same power of two (\( x_i \) and \( y_i \)) modulo two, that we will represent by \( x_i \oplus y_i \) (usually called **XOR**), and taking into account the carry bit \( c_i \), as shown in table 1. If we take into account the carry bit \( c_{i-1} \) due to the addition of the two previous bits \( x_{i-1} \) and \( y_{i-1} \), the adder will have three inputs and a full addition (**full adder**) will be carried out, as shown in table 2.

<table>
<thead>
<tr>
<th>( x_i )</th>
<th>( y_i )</th>
<th>( c_{i-1} )</th>
<th>( x_i \oplus y_i \oplus c_{i-1} )</th>
<th>( c_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
The efficient way of working out operations with positive and negative numbers in classical Boolean arithmetics is to use the 2’s-complement representation of the number, which allows us to work with the sign bit in the same way we do with any other bit. Given a Boolean number \( p \) of the form (1), its 2’s-complement representation \( [8, 9] \) is

\[ C_2(p) = 2^n - p. \]

(2)

That information representation will allow us to design simpler digital quantum circuits and will also avoid fundamental problems, such as having different ‘positive’ and ‘negative’ zeros. Signed integer multiplication is usually (classically) performed encoding the multiplier and performing additions, subtractions and bit displacements using the so-called Booth’s algorithm [8]. The main idea of this algorithm, which will be analyzed later on, is to represent binary data as unsigned integers in order to perform additions. This can be achieved by the 2’s-complement representation, that makes possible performing subtraction by means of addition [9]. With this idea in mind, we can work out additions and subtractions in a very easy way, and tackle the multiplication of signed integers. For this purpose, which is not at all a trivial arithmetic operation from the point of view of computer science, a quantum Booth multiplier (QBM) is presented based on the corresponding classical Booth multiplication algorithm, which is briefly described in the next section.

3. The classical Booth algorithm

The classical Booth multiplication algorithm, as a first step, encodes binary chains by means of their transitions between 0’s and 1’s as shown in Figure 1. Once these transitions are detected, the necessary displacements and additions are performed. This scheme extends to any number of blocks of 1’s in a multiplier, including the case in which a single 1 is treated as a block. Indeed, we might say that the classical Booth’s algorithm detects ‘islands’ of 1’s.

\[ 0 0 1 0 1 1 1 0 1 1 0 1 0 0 1 0 \quad 0 1 1 1 0 0 1 1 0 1 1 0 1 1 \]

Figure 1. On the right part of the figure is Booth’s encoding of the 16 bits chain shown on the left. Transitions 0 \( \rightarrow \) 1 and 1 \( \rightarrow \) 0 are represented by 1 and \( \bar{1} \), respectively.

It is well known that multiplication can be achieved by adding appropriately shifted copies of the multiplicand and that the number of such operations in a block of 1’s can be reduced to two if we observe that

\[ 2^n + 2^{n-1} + \cdots + 2^{n-k} = 2^{n+1} - 2^{n-k}. \]

(3)

Therefore, the algorithm encodes the binary alphabet \( \{1, 0\} \) in another one, \( \{0, 1, -1\} \), that shows the transitions among the binary logic states within a binary chain. Thus, when a \( -1 \) appears, a subtraction (using the 2’s-complement representation) and a shift are performed, and when a 1 appears an addition and a shift are performed. In the other two cases, just an arithmetic shift is performed for each situation. The algorithm conforms to this scheme by performing a subtraction when the first 1 of the block is encountered (1 \( \rightarrow \) 0) and an addition when the end of the block is encountered (0 \( \rightarrow \) 1). It is straightforward to show that the same scheme works for a negative multiplier using the 2’s-complement notation.

The architecture of Booth’s algorithm, just described, is shown in Fig. 2, where the multiplier and the multiplicand are placed in the \( Q \) and \( M \) registers, respectively; there is also a 1-bit
register placed logically to the right of the Less Significant Bit (LSB) $Q_0$ of the $Q$ register and designated $Q_{-1}$; finally, the result of the multiplication is loaded in the $A$ and $Q$ registers, the first one initialized to 0, and is read from the Most Significant Bit (MSB) on the left to the LSB on the right in the juxtaposition of the registers $A$ and $Q$.

![Classical multiplier architecture.](image)

Control logic scans the bits of $Q_0$ and $Q_{-1}$ one at a time. According to Fig. 3, if the two bits are the same, then all the bits of $A$, $Q$, and $Q_{-1}$ are shifted to the right one bit. If the two bits differ, then the multiplicand is added or subtracted from the $A$ register, according as the two bits are changing $0 \rightarrow 1$ or $1 \rightarrow 0$. Following the addition or subtraction, the right shift occurs. In either case, the right shift is such that the leftmost bit of $A$, namely $A_{n-1}$, not only is shifted into $A_{n-2}$, but also remains in $A_{n-1}$. This is required to preserve the sign of the number in $A$ and $Q$ and it is named *arithmetic shift* [8], since it preserves the sign bit.

4. The quantum Booth algorithm

Now that the classical Booth algorithm is well understood, our version of the quantum Booth’s algorithm will be presented. First of all, we have to represent the new alphabet $\{0, 1, -1\}$ by three orthonormal quantum states belonging to the basis of $\mathcal{H}^2 \otimes \mathcal{H}^2$ as follows:

$$|00\rangle \equiv 0, \quad |01\rangle \equiv 1, \quad |10\rangle \equiv -1.$$  \hspace{1cm} (4)

They have been chosen in this way because they just differ from each other in one binary digit, a useful property that will simplify the control stage in the QBM architecture. To encode two adjacent qubits $|x_{i+1}\rangle$ and $|x_i\rangle$ an ancillary $|0\rangle$ is needed, and therefore the number of qubits required for that is three, which will be denoted as

$$|x_{i+1}\rangle_1, \quad |0\rangle_2 \quad \text{and} \quad |x_i\rangle_3.$$  \hspace{1cm} (5)

They are transformed into

$$|x_{i+1}\rangle_1, \quad |x'_{i+1, i}\rangle_2 \quad \text{and} \quad |x''_{i+1, i}\rangle_3.$$  \hspace{1cm} (6)
respectively, by means of the circuit shown in Figure 4, that encodes all the input possibilities according to the rules shown in Figure 5, which involves CNOT and CSWAP gates. From now on, we will denote by \( \mathcal{B}E \) this Booth encoding procedure.

\[
\begin{align*}
|x_i+1\rangle_1 & \quad |x_i+1\rangle_2 \\
|0\rangle_2 & \quad |x'_i+1\rangle_2 \\
|x_i\rangle_3 & \quad |x'_i\rangle_3 \\
\end{align*}
\]

Figure 3. Flow chart for classical Booth’s algorithm.

Figure 4. Booth’s encoding of two adjacent qubits.

Figure 5. Transforming the state of two adjacent qubits.

An example showing how a 4-qubit \( \mathcal{B}E \) encodes is presented in Figure 6. Notice that for reversibility purposes the inverse of the circuit has to be applied, \( \mathcal{B}E^{-1} \), which is the same circuit as \( \mathcal{B}E \) but uses the outputs as inputs and viceversa.

Following this 4-qubit example, let us consider now the multiplication of two 4-qubit numbers represented by the multiplier ket \( |x_3, x_2, x_1, x_0\rangle \) and the multiplicand ket \( |y_3, y_2, y_1, y_0\rangle \).
multiplier is encoded by the Booth encoder circuit and controls the logic operations on the multiplicand that load the proper partial products in their registers. The product (the final output) will have 8 qubits (the same length that will have the partial product registers) and is given in 2’s-complement representation. As it is shown in Fig. 7 the partial products are arranged in order to reproduce the multiplication inherent displacements. In summary, the quantum circuit shown in Fig. 6, encodes the binary transitions \((i+1, i)\) and controls, by Toffoli gates, what will be stored at the quantum registers, where the partial products have been loaded. Hence, depending on the transitions in the quantum binary chain, we will perform one of the three following operations over the multiplicand:

- If there is no transition, \(|00\rangle\), then 0’s are loaded in the partial product registers.
- If there are up-transitions, \(|01\rangle\), then the multiplicand is loaded in the partial product registers.
- If there are down-transitions, \(|10\rangle\), then the multiplicand 1’s-complement representation is loaded in the partial product registers.

Finally, we have to sum all the partial products that we have generated. If the multiplicand and the multiplier have \(n\) qubits, each partial product will have \(2n\) qubits and they will be shifted, from the encoded multiplier LSB to the MSB, depending on the ordinal position of the multipliers bits. These displacements are implemented within the quantum \(2n\)-registers hardware.

After the \(O(\log n)\) sums have been performed we still have to do something more because we have used 1’s-complement representation instead of 2’s-complement representation. Indeed, we have saved the “carry” bit bearing in mind that 2’s-complement representation is equal to 1’s-complement representation plus one. Therefore, we have to add, to the whole partial products sum, another register where the 1’s have been stored in their proper ordinal way, as one can
see in the example displayed in Fig. 7. Therefore, the final output is the desired input product represented in 2's-complement notation.

As an example of how the QBM circuit shown in Fig. 7 works, let us trace now the fourth partial product \( S_3 \) in Figure 7 of the multiplication of \( M = 0010 \) by \( Q = 0110 \) and \( M' = 1110 \) by \( Q' = 0110 \), where \( 6_{10} \equiv 10 - 10 \) is the codification of \( 6_{10} \) by Booth’s circuit encoder. For the sake of simplicity the multiplier is the same for both examples, which are explicitly shown in Figure 8.

From the quantum circuit point of view the fourth partial product (for both examples) is the instantiation of the quantum register shown in Fig. 9 with the notation

\[
| y_3, y_2, y_1, y_0 \rangle \equiv | y_{(4)} \rangle. \tag{7}
\]

Then, the circuit adds, using the most efficient quantum adders that already exist [5], all the partial products that, eventually, give us the result called \( S_{Tot} \). For achieving the final result we still have to add the content of the quantum register \( C_2 \) where the proper value for working out the 2's-complement operation has been stored. In that example the value loaded in \( C_2 \) is

\[
\sum \left( S_0 + S_1 \right) = S_{Tot}.
\]
Figure 8. Two examples of multiplication using Booth’s encoding.

\[
\begin{array}{cccc}
2_2) & 0 & 0 & 1 \\
6_2) & \times & 1 & 0 & -1 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
\end{array}
\]

00000010, giving us, as a final and correct result, 00001100 for the multiplication of \(2_2) \times 6_2)\) and 11110100 (the 2’s complement of 00001100) for the multiplication of \(-2_2) \times 6_2)\).

Figure 9. Quantum registers instantiation

\[
\begin{array}{cccc}
|0\rangle \leftarrow 0 & |0\rangle \leftarrow 1 \\
|\hat{y}(4)\rangle \leftarrow 0010 & |\hat{y}(4)\rangle \leftarrow 1110 \\
|\hat{0}(3)\rangle \leftarrow 000 & |\hat{0}(3)\rangle \leftarrow 000 \\
|\hat{0}(2)\rangle \leftarrow 00 & |\hat{0}(2)\rangle \leftarrow 000 \\
|\hat{y}(4)\rangle \leftarrow 0000 & |\hat{y}(4)\rangle \leftarrow 0000 \\
|\hat{0}(3)\rangle \leftarrow 111 & |\hat{0}(3)\rangle \leftarrow 000 \\
|\hat{y}(4)\rangle \leftarrow 1101 & |\hat{y}(4)\rangle \leftarrow 0001 \\
|0\rangle \leftarrow 0 & |0\rangle \leftarrow 0 \\
|\hat{0}(4)\rangle \leftarrow 0000 & |\hat{0}(4)\rangle \leftarrow 0000 \\
|\hat{y}(4)\rangle \leftarrow 0000 & |\hat{y}(4)\rangle \leftarrow 0000 \\
\end{array}
\]

5. Discussion and proposed future directions

It is important to notice that the QBM has a very regular structure. It can be easily implemented in order to build bigger circuits for bigger inputs than the one shown in Fig. 7. Its delay is \(O((\log n)^2)\) due to the fact that the adders structure is \(O(\log n)\) and that each adder has a logarithmic depth. One could reduce it by means of a CSA scheme [10], but it would then destroy the circuit’s regular structure, which is an undesirable compensation. Thus, the bottlenecks are the adders and our future research efforts will go in the direction of clarifying whether \(O(\log n)\) is a fundamental bound, from the quantum physics point of view, or not.

Note that adder information processing [6] is classical because the input information is also classical; there is neither entanglement nor quantum superposition. Therefore, the Quantum Booth Algorithm architecture presented here is also a classical reversible one, as a consequence of following the research direction pointed by [4, 5, 6], and using only TOFFOLI, SWAP and CNOT, and no Hadamard gates. Indeed, what we have accomplished here is interesting by itself (to our knowledge, there are no quantum circuits for multiplying signed integers), but if we want to go a step further towards designing faster quantum circuits, it is necessary to reconsider the
way in which the quantum arithmetic circuits have been designed. It is essential to take into account the intrinsic quantum ingredients: entanglement and superposition of states. Work in this direction is in progress.

Acknowledgments
The authors thank Profs. C Gómez, G Brassard, J M Fernández and M L Glasser for valuable discussions and advice. This work has been partially supported by Spanish Ministerio de Educación y Ciencia (Project MTM2005-09183) and Junta de Castilla y León (Excellence Project VA013C05). The Q-circuit package was used to produce the quantum circuits included in this paper.

References